University of West Attica & University of Limoges

Artificial Intelligence & Visual Computing

Academic Year 2024 – 2025

Department**: Information Technology and Computer Engineering**

Report: **Steam Reviews Sentiment Analysis**

Author: **Nikolaos Theokritos Tsopanidis**

StudenID: **24022**

Course Title: **Natural Language Processing**

Date of Submission: **17/01/2025**

Contents

Introduction

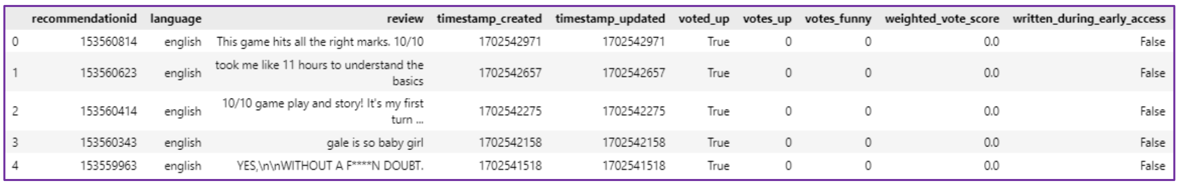
For this sentiment analysis task, I selected a dataset containing approximately 309.103 user reviews of Baldur's Gate 3 from the Steam platform, updated up to 14 December 2023. The dataset includes reviews for both the early access and full release versions of the game, with all reviews written in English. The data was obtained using Steam's API.[1]

Why This Dataset

Baldur's Gate 3, released in 2023, is a role-playing game created and published by Larian Studios, a game studio from Belgium. The game has caught the attention of players all over the world and it is a perfect candidate for sentiment analysis. By digging into user reviews, we can uncover how players really feel about the game and what they think about its different features.

Dataset Description

The initial dataset consists of **13 columns**:



	recommendationid	language	review	timestamp_created	timestamp_updated	voted_up	votes_up	votes_funny	weighted_vote_score	written_during_early_access
0	153560814	english	This game hits all the right marks. 10/10	1702542971	1702542971	True	0	0	0.0	False
1	153560623	english	took me like 11 hours to understand the basics	1702542657	1702542657	True	0	0	0.0	False
2	153560414	english	10/10 game play and story! It's my first turn ...	1702542275	1702542275	True	0	0	0.0	False
3	153560343	english	gale is so baby girl	1702542158	1702542158	True	0	0	0.0	False
4	153559963	english	YES,\n\nWITHOUT A F****N DOUBT.	1702541518	1702541518	True	0	0	0.0	False

1. **recommendationid**: Unique id of each recommendation.
2. **language**: Review's written language.
3. **review**: The text of the written review.
4. **timestamp_created**: Date the review was created (in Unix Timestamp format).
5. **timestamp_updated**: Date the review was last updated (in Unix Timestamp format).
6. **voted_up**: A binary indicator where true means the review is a positive recommendation.
7. **votes_up**: How many Steam users found a particular review helpful.
8. **votes_funny**: How many Steam users found a particular review funny.
9. **weighted_score_vote**: Score of helpfulness.
10. **written_during_early_access**: Whether the user created the review while the game was in early access (if the game was purchased in pre-release development stage).

However, for this analysis, I selected only the columns "**review**" and "**voted_up**".

By focusing on these columns, the dataset is optimized for the sentiment classification task, where the goal is to categorize user reviews as positive or negative based on the "**voted_up**" field.

Preprocessing the Dataset

The entire preprocessing task was executed in a **Jupyter Notebook** environment using the **pandas** library. The preprocessing steps are outlined below.

Data Import and Initial Exploration

The dataset was loaded, and the first five rows were displayed to understand its structure. **Exploratory Data Analysis (EDA)** was conducted to gather information on the dataset's size, data types, and column descriptions. Key observations included **309.103** total entries, **13** total columns and **22,4 MB** size of data.

Imbalance in Data

The dataset contained **298.932 ‘True’** values (positive reviews) and **10.171 ‘False’** values (negative reviews) in the “**voted_up**” column. This imbalance reflects the game's popularity and overall success. However, every negative review provides valuable insights for potential improvements.

Handling Missing Values

There were **1,017 missing values** in the review column. These rows were dropped to ensure data quality.

Binary Labeling

The “**voted_up**” column was transformed into **binary numerical values** (‘**1**’ for True and ‘**0**’ for False) for ease of calculations.

Balancing the Dataset

To address the imbalance, the dataset was resampled to include an equal number of positive and negative reviews. The final dataset contained **10.157** positive reviews and **10.157** negative reviews, achieving perfect balance. This step was crucial to avoid bias in machine learning algorithms towards the overrepresented positive reviews.

Text Data Processing

- **Final Dataset Size**

  The filtered and balanced dataset now consists of **20,314** entries and is approximately **317,5 KB** in size.

- **Removing Special Characters**

  A custom function, `remove_special_characters()`, was defined to remove special characters and numbers, which can introduce noise to the analysis. Additionally, all text was converted to lowercase to reduce the number of unique words.

- **Tokenization**

  Using the `ToktokVectorizer()` from the **nltk** library, the text was tokenized. This tool provides a simple and general tokenizer that is particularly suitable for this dataset.

- **Stopword Removal**

  Stopwords were removed using the English stopwords list from **nltk**, while retaining *"not"* and *"no"* due to their importance in sentiment context. A new column “**without_stopwords**” was created to store the text data without stopwords, while retaining the original text for reference.

- **Lemmatization**

  The `WordNetLemmatizer()` from **nltk** was employed to lemmatize the text data, converting each word to its base form. The lemmatized data was saved in another separate column named “**lemmatized**”.

The differences between the original, stopword-removed, and lemmatized text were displayed by printing the first row of each format for comparison.

Original text sample	['dumb', 'ally', 'ai', 'who', 'likes', 'to', 'kill', 'themselves', 'proccing', 'opportunity', 'attacks', 'to', 'leave', 'just', 'to', 'misty', 'step', 'back', 'into', 'the', 'middle', 'of', 'enemies', 'patches', 'that', 'introduce', 'more', 'bugs', 'than', 'fix']
Text sample without stopwords	['dumb', 'ally', 'ai', 'likes', 'kill', 'proccing', 'opportunity', 'attacks', 'leave', 'misty', 'step', 'back', 'middle', 'enemies', 'patches', 'introduce', 'bugs', 'fix']
Text sample lemmatized	['dumb', 'ally', 'ai', 'like', 'kill', 'proccing', 'opportunity', 'attack', 'leave', 'misty', 'step', 'back', 'middle', 'enemy', 'patch', 'introduce', 'bug', 'fix']

Analyzing the printed rows, words such as *“who”, “themselves”, ”to”* etc. were characterized stop words and were removed from text. The lemmatized row also displays some differences in the words *“likes”, “attacks”, “patches”, “enemies”* and *“bugs”* where they were converted into their base forms.

Saving the Processed Dataset

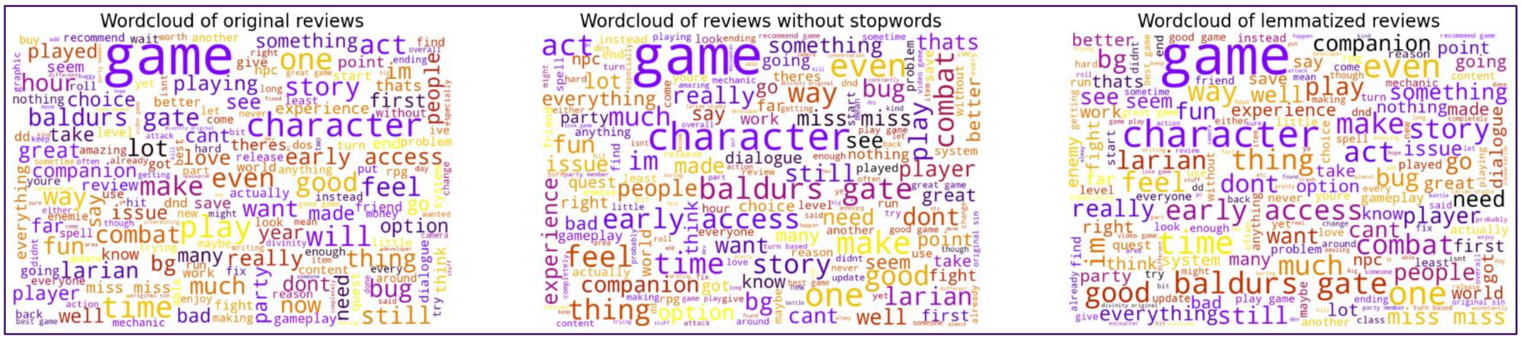
The fully processed dataset, with the additional columns for cleaned and lemmatized text, was saved as a .CSV file named “**BG3_Cleaned.csv**”. This file serves as the foundation for subsequent analysis and modeling tasks.

Visualizing the Text Data with Word Clouds

To gain a better understanding of the textual content, word clouds were created to highlight the most representative words in the dataset.

Word Clouds for Each Review Form

Word clouds were generated for the original text data, the text data without stopwords, and the lemmatized text data. Upon visual inspection, no significant differences were discerned between the three word clouds.

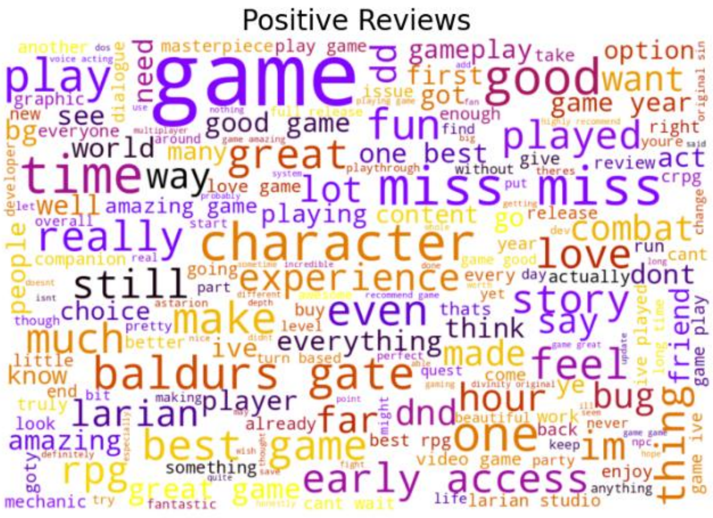


Word Clouds for Positive and Negative Reviews

Separate word clouds were created from the text data without stopwords for positive and negative reviews. The exclusion of stopwords ensured a focus on sentiment-bearing words.

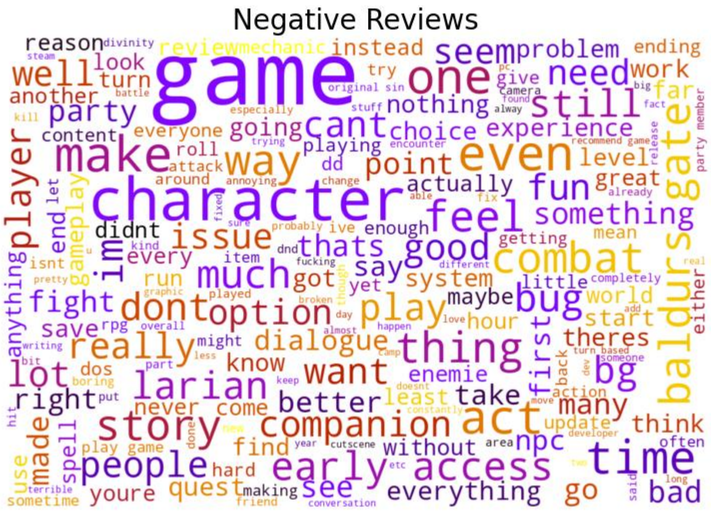
- Positive Reviews

The word cloud prominently featured words such *as* "*best*", "*good*", "*fun*", "*great*", "*amazing*".



- Negative Reviews

Words like "*issue*", "*bug*", "*problem*" were observed.



Interestingly, some words like "*good*", and "*bug*" appeared in both word clouds, reflecting their dual relevance across sentiments.

Insights from Word Clouds

While the word clouds provided an overview of commonly used terms, certain nuances, such as sentiments expressed through multi-word phrases, were not captured. This observation suggested that a bag-of-words model might offer deeper insights by considering word combinations.

Creating Training, Validation, and Test Datasets

To prepare the dataset for machine learning, the `train_test_split()` function from the "scikit-learn" library was used to split the data into training, validation, and test datasets. This process was applied to all three forms of text data (original, without stopwords, and lemmatized).

Dataset Splitting

The dataset was first split into training (**80%**) and test (**20%**) sets. The training set was further split, with **10%** of the training data reserved for validation and the remaining used for training.

Feature Extraction

- Bag of Words

  Using the `CountVectorizer()` function from **scikit-learn**, Bag of Words representations were created to capture the frequency of unigrams, bigrams, and trigrams for each text data form.

- TF-IDF

  The `TfidfVectorizer()` function from **scikit-learn** was used to create Term Frequency-Inverse Document Frequency (**TF-IDF**) features. This method emphasizes terms that are significant in individual documents relative to the entire corpus.

Purpose of Feature Extraction

These features were generated to compare the performance of machine learning models trained on different combinations of data representations. Original data with Bag of Words or TF-IDF features, data without stopwords with Bag of Words or TF-IDF features and lemmatized data with Bag of Words or TF-IDF features.

By completing these steps, the dataset was fully prepared for model training and evaluation, ensuring foundation for comparing the efficiency of various machine learning models in predicting sentiments.

Machine Learning Models for Sentiment Analysis

For the sentiment analysis and classification task, I employed three machine learning models from the **scikit-learn** library.

Logistic Regression

A widely used linear model for binary classification tasks. It predicts the probability of occurrence of a binary dependent variable, performing calculations utilizing a logit function.

SGD Classifier

Utilizes Stochastic Gradient Descent (**SGD**), which is an efficient method for training linear classifiers and regressors. The default loss function used was hinge, making it effectively equivalent to a linear Support Vector Machine (**SVM**).

Multinomial Naive Bayes

A probabilistic classifier that assumes a feature vector where each element represents the frequency of a term. This model is particularly effective for text classification tasks[2]. Gaussian Naive Bayes was also considered, which assumes continuous features[3] characterized by a Gaussian distribution. However, it is less suited for this dataset's discrete feature structure.

Deep Learning Models for Sentiment Analysis

In addition to traditional ML approaches, three Deep Learning models were implemented using **TensorFlow** and **Keras** frameworks.

Simple Recurrent Neural Network (RNN)

A basic sequential neural network architecture designed for processing text data. The implementation of the RNN model was guided by practical examples from GeeksforGeeks.[4]

Long Short-Term Memory Network (LSTM)

A variant of RNNs capable of capturing long-term dependencies within sequential data, making it particularly effective for sentiment analysis. This approach followed the guidelines from Castillo (2023) [5].

Convolutional Neural Network (CNN)

Although traditionally used for image processing, CNNs have shown strong performance in extracting patterns from textual data. The implementation of CNNs for text classification was based on Brownlee (2017) [6], which demonstrated how convolutional layers can be applied effectively in sentiment analysis tasks.

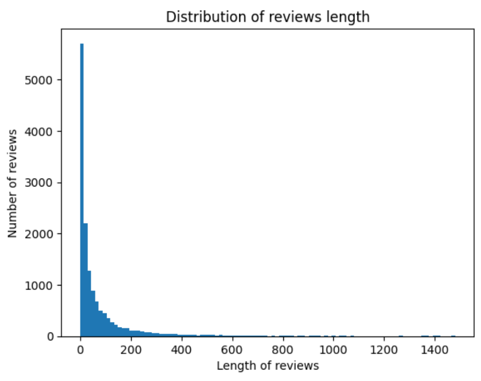
Text Preprocessing for Neural Networks

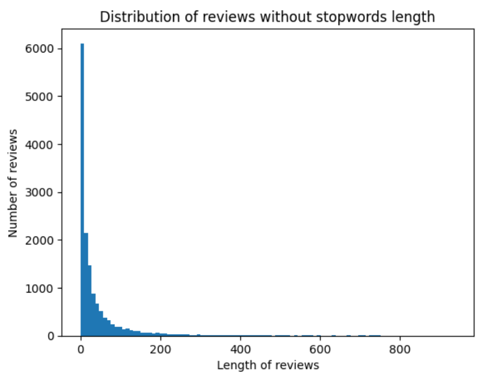
Tokenization

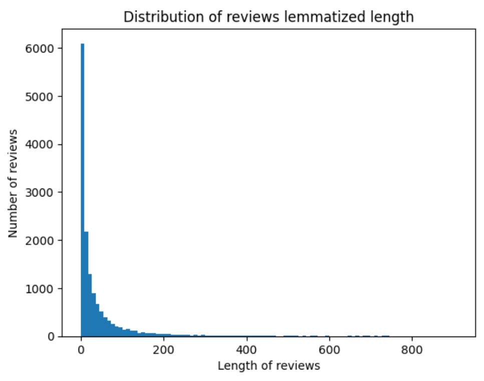
TensorFlow's tokenizer was used to tokenize the text data, as indicated from GeeksforGeeks [7]. I extracted the vocabulary sizes for each text form. The original data includes **27.701** unique tokens, the data without stopwords **27.566** and the lemmatized **25.096**. The vocabulary sizes will be used as attributes in the Embedding layers. A `text_to_sequences` function will then convert tokens into sequences for each text form.

Maximum Sequence Length & Padding

Visualizations showing distributions of review lengths were created to define the maximum length each sequence will take, defining a fixed size input for the neural networks to receive. I chose maximum lengths for each text form to represent most reviews, excluding outlier sequences with great lengths. I set for the original text the maximum length of **400**, **200** for lemmatized data and **200** for data without stopwords. The sequences were padded using these values, to ensure fixed input sizes.







Training and Validation

Model Training

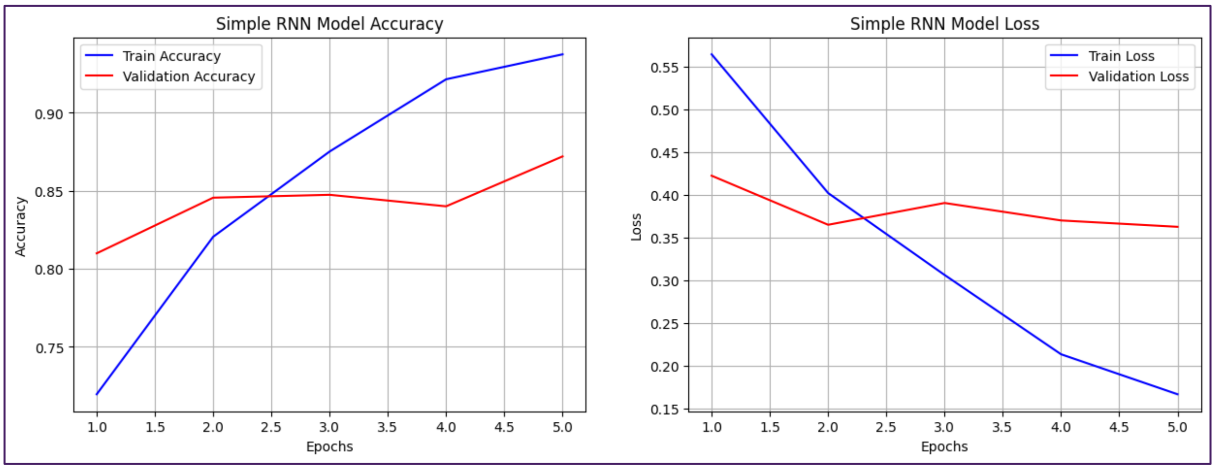
Each model was trained on the respective dataset combinations. For the neural networks, as loss function I defined the binary cross entropy, for optimizer “**Adam**” and selected “**accuracy**” as evaluation metric.

Validation

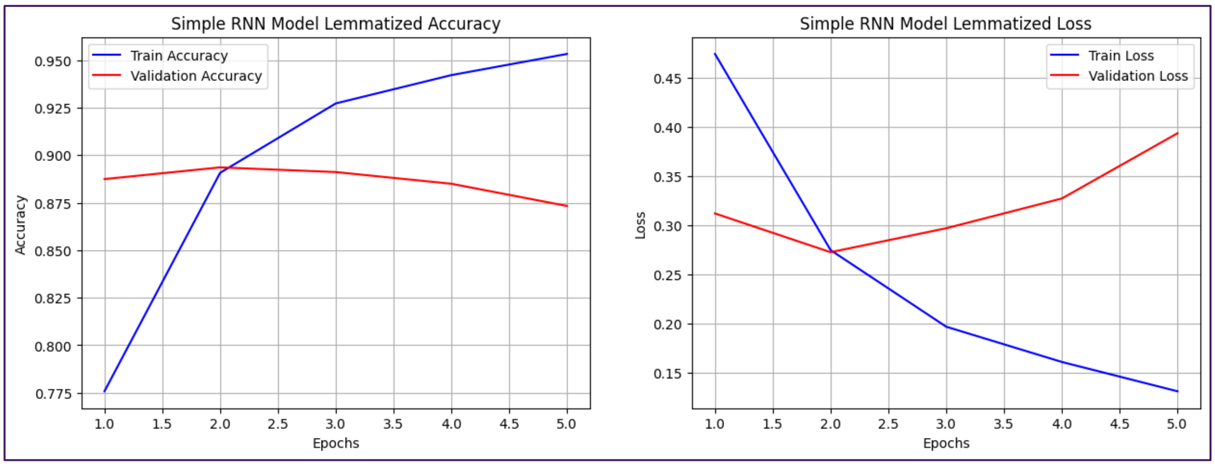
To validate the performance of each Machine Learning model, the `cross_val_score()` function from **scikit-learn** was used. Cross-validation was performed using the validation data splits created earlier during data preprocessing. Deep Learning models were validated using the validation sets over **5** epochs.

Neural Network Training Visualizations

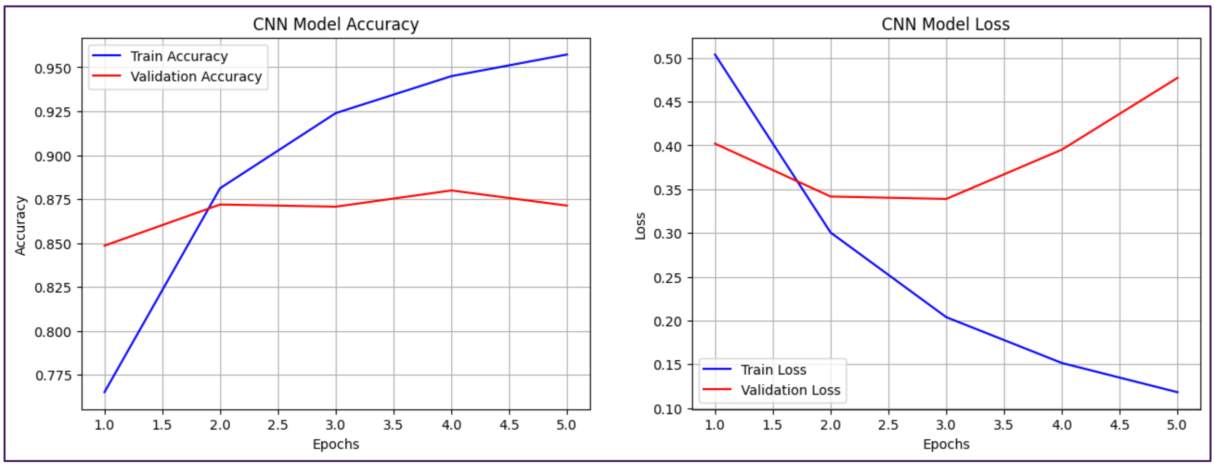
During the training process of each neural network (Simple RNN, LSTM, and CNN), accuracy and loss values were visualized across epochs. These visualizations included both training and validation accuracy and loss metrics, providing insights into model performance and overfitting trends.



For the Simple RNN model, the training accuracy increased steadily across epochs, reaching above 90% by the 4th epoch. Training loss decreased consistently, suggesting the model learned effectively from the data. Validation accuracy showed a stable increase with no significant divergence from training accuracy. Validation loss also decreased, reflecting no overfitting or underfitting. The Simple RNN training was successful and showed balanced performance between training and validation datasets.



The LSTM's training accuracy increased from 0.77 to 0.95, indicating sufficient learning progress. Loss decreased progressively with epochs reaching ~0.12. Validation accuracy remained stuck around 0.87, showing a possible overfitting. Validation loss was increasing after the 2nd epoch indicating possible overfitting. The LSTM model showed a decent efficiency during the training process, but the increasing validation loss could mean that there is a need for better tuning.



For CNN, the training accuracy improved steadily with each epoch, crossing 95% by the 5th epoch. Training loss decreased consistently, reflecting efficient learning. Validation accuracy saturated around 0.875 after the 3rd epoch, showing the model's ability to generalize. Validation loss initially decreased but started increasing after the 3rd epoch, indicating slight overfitting. The CNN model performed well, but the increasing validation loss after the 3rd epoch suggests some overfitting.

Model Evaluation

To evaluate the performance of both the machine learning and deep learning models, several metrics were utilized.

- Accuracy Score

The `predict` function was used to generate predictions, and the accuracy score was calculated based on the outputs. The accuracy score is calculated by dividing the number of correct predictions by the total number of predictions.

- Classification Reports

Included metrics such as precision, recall, and F1 score for a more comprehensive analysis of model performance. Precision measures how often the model predicts a positive class correctly. It is calculated by dividing the True Positive Predictions by the sum of the True Positives with the False Positives. Recall measures if the model identifies the positive samples correctly as positive and not as negative. It is calculated by dividing the True Positives by the sum of True Positives with False Negatives. The F1 score represents both precision and recall. It is calculated by dividing the product of Precision and Recall by the sum of Precision and Recall, multiplied by 2. A perfect F1 score is 1, which indicates a perfect model, and the lowest possible values is 0.

- Confusion Matrices

Visualized to better understand the distribution of True Positives, True Negatives, False Positives, and False Negatives.

▪ ROC Curve

Used to analyze model efficiency across a range of thresholds. Designed by plotting the True Positive Rate, often referred to as Sensitivity or Recall, against the False Positive Rate, which is the complement of Specificity. AUC Score, or area under the curve, is measured and describes the model's ability to correctly distinguish positive and negative samples. The perfect score is 1 and the lowest, which corresponds to random guessing, is 0.5.

▪ Computational Time

Recorded during the training process to judge the models based on their speed.

Results

Below is the table for summarizing the results of model evaluations:

Model Type	Data Form	Feature Type	Accuracy	Precision		Recall		F1 Score		AUC (ROC)
				Neg.	Pos.	Neg.	Pos.	Neg.	Pos.	
Logistic Regression	Original	Bag of Words	0.89	0.92	0.87	0.86	0.93	0.89	0.9	0.95
		TF-IDF	0.88	0.86	0.91	0.91	0.85	0.89	0.88	0.88
	Without Stopwords	Bag of Words	0.89	0.92	0.86	0.85	0.92	0.88	0.89	0.95
		TF-IDF	0.88	0.85	0.91	0.92	0.84	0.89	0.88	0.95
	Lemmatized	Bag of Words	0.89	0.91	0.86	0.85	0.92	0.88	0.89	0.95
		TF-IDF	0.88	0.85	0.92	0.92	0.84	0.89	0.88	0.95
SGD Classifier	Original	Bag of Words	0.89	0.9	0.89	0.88	0.9	0.89	0.9	0.95
		TF-IDF	0.9	0.89	0.92	0.92	0.89	0.91	0.9	0.97
	Without Stopwords	Bag of Words	0.89	0.9	0.88	0.88	0.91	0.89	0.89	0.95
		TF-IDF	0.9	0.88	0.92	0.93	0.87	0.9	0.9	0.96
	Lemmatized	Bag of Words	0.89	0.9	0.89	0.89	0.9	0.89	0.89	0.95
		TF-IDF	0.9	0.87	0.92	0.93	0.87	0.9	0.89	0.96
Multinomial Naïve Bayes	Original	Bag of Words	0.75	0.67	0.98	0.99	0.52	0.8	0.68	0.9
		TF-IDF	0.76	0.68	0.98	0.99	0.54	0.81	0.7	0.87
	Without Stopwords	Bag of Words	0.73	0.65	0.98	0.99	0.48	0.79	0.64	0.91
		TF-IDF	0.8	0.72	0.97	0.98	0.62	0.83	0.76	0.94
	Lemmatized	Bag of Words	0.75	0.66	0.98	0.99	0.51	0.8	0.67	0.91
		TF-IDF	0.8	0.72	0.97	0.98	0.62	0.83	0.75	0.94
Simple RNN	Original	Tokenized & padded	0.84	0.82	0.86	0.86	0.82	0.84	0.84	0.91
	Without Stopwords		0.77	0.76	0.78	0.78	0.76	0.77	0.77	0.85
	Lemmatized		0.86	0.89	0.83	0.81	0.9	0.85	0.86	0.92
LSTM	Original	Tokenized & padded	0.88	0.85	0.92	0.93	0.84	0.89	0.88	0.88
	Without Stopwords		0.87	0.88	0.87	0.86	0.88	0.87	0.88	0.87
	Lemmatized		0.87	0.85	0.91	0.91	0.83	0.88	0.87	0.87
CNN	Original	Tokenized & padded	0.9	0.86	0.86	0.85	0.9	0.87	0.88	0.94
	Without Stopwords		0.86	0.87	0.86	0.86	0.87	0.86	0.87	0.92
	Lemmatized		0.86	0.85	0.87	0.87	0.85	0.86	0.86	0.92

* ***Neg***. *: Negative Class* | ***Pos***. *: Positive Class*

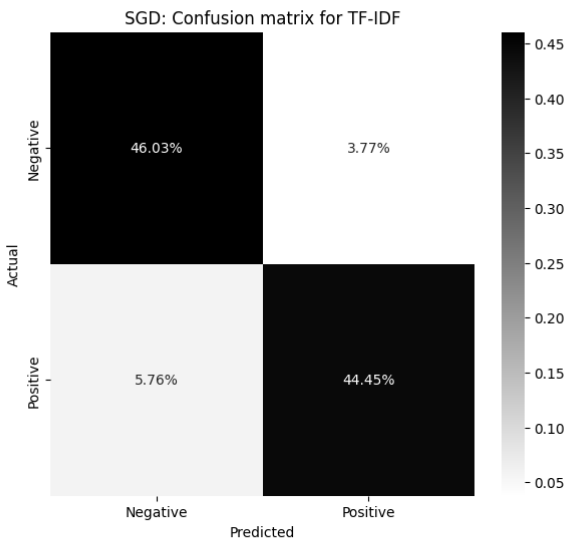
The next table displays the computational time each model took:

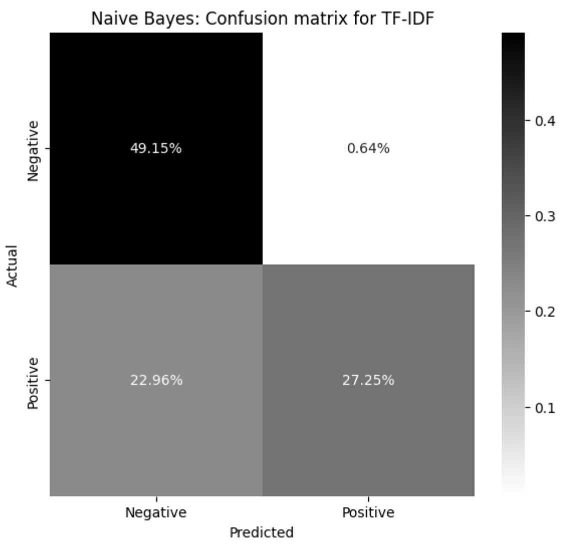
Model Type	**Data Form**	**Feature Type**	**Time (seconds)**
Logistic Regression	Original	Bag of Words	8.97
		TF-IDF	2.39
	Without Stopwords	Bag of Words	5.37
		TF-IDF	2.88
	Lemmatized	Bag of Words	5.43
		TF-IDF	2.61
SGD Classifier	Original	Bag of Words	0.31
		TF-IDF	0.12
	Without Stopwords	Bag of Words	0.11
		TF-IDF	0.09
	Lemmatized	Bag of Words	0.12
		TF-IDF	0.08
Multinomial Naïve Bayes	Original	Bag of Words	0.05
		TF-IDF	0.04
	Without Stopwords	Bag of Words	0.03
		TF-IDF	0.03
	Lemmatized	Bag of Words	0.02
		TF-IDF	0.03
Simple RNN	Original	Tokenized & padded	35.75
	Without Stopwords		18.65
	Lemmatized		20.52
LSTM	Original	Tokenized & padded	147.37
	Without Stopwords		86
	Lemmatized		97.3
CNN	Original	Tokenized & padded	5.01
	Without Stopwords		4.64
	Lemmatized		4.4

Visualizations

For the sake of brevity, since there are many examples, only the confusion matrices of the models with the best and worst results have been attached on this report.

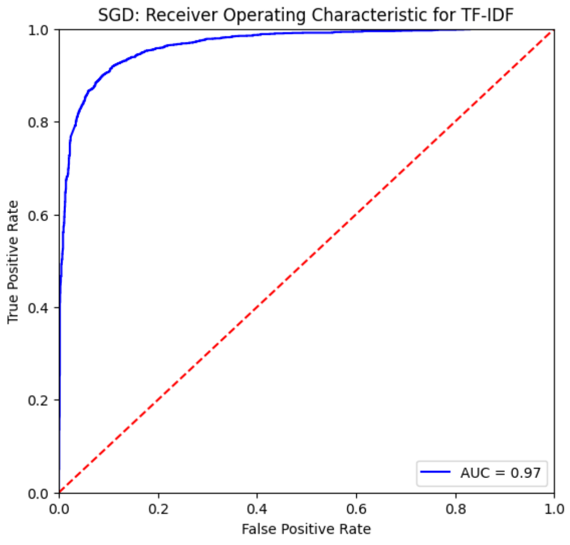
Confusion Matrices

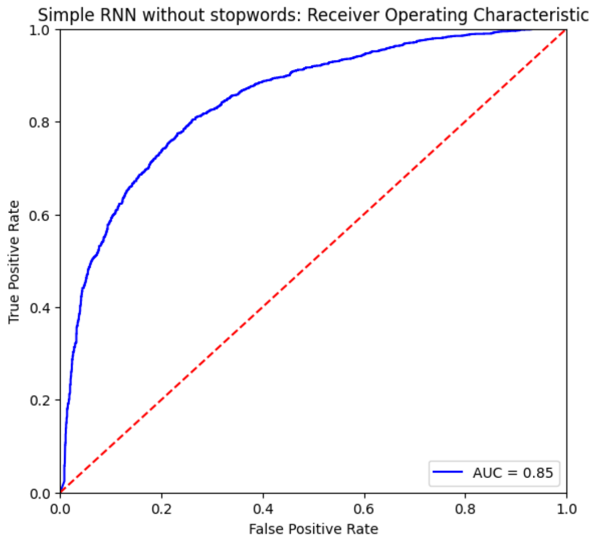




The SGD Classifier achieved almost a perfect score predicting the testing sentiments to their corresponding classes, while the Multinomial Naïve Bayes model had poor results ranking the data, categorizing a large proportion of positive data as negatives, indicating a bias towards the negative class.

ROC Curves





The ROC curve and the AUC score (0.97) represent the probability that the SGD Classifier will perform the best at categorizing a review correctly. The Simple RNN model's ROC curve tends less toward the upper left edge of the graph, and the AUC score is 0.85, implying that there is less probability of the model to rank correctly a review sample.

Results Analysis

- Logistic Regression

Logistic Regression consistently performed well across different preprocessing methods and feature types. The best accuracy (0.89) was observed across the settings with Bag of Words. F1-scores for both classes were consistently high, ranging between 0.88 and 0.9 for Bag of Words, and 0.88 to 0.89 for TF-IDF. Lemmatization and removal of stopwords had no significant impact on its performance. The AUC-ROC remained stable at 0.95 for Bag of Words and 0.89 for TF-IDF.

- SGD Classifier

This model demonstrated slightly better results compared to Logistic Regression, especially with TF-IDF. Best accuracy (0.9) observed in data with TF-IDF. TF-IDF showed superior performance, with an AUC-ROC of 0.97, higher than Bag of Words. F1-scores were balanced across negative and positive classes, ranging from 0.89 to 0.91 depending on the preprocessing method and feature type. Removing stopwords or lemmatization didn't degrade performance, maintaining consistent accuracy and AUC-ROC.

- Multinomial Naïve Bayes

This model struggled in comparison to others but showed moderate performance in general. Best accuracy was observed in data with TF-IDF features. High recall for the negative class (0.98 or above) was a consistent strength, although precision was significantly lower (~0.67). Accordingly, high precision for the positive class (~0.98) was detected as opposed to recall (~0.55). The AUC-ROC varied between 0.87 and 0.94, with the highest value observed in TF-IDF features.

- Simple RNN

The Simple RNN model had moderate success with lower accuracy than Logistic Regression or SGD Classifier and slightly above Multinomial Naïve Bayes. The best accuracy (0.86) was detected in lemmatized data. Lemmatization slightly improved precision for negative class and recall for the positive class. Without stopwords, the performance dipped slightly in all metrics.

- LSTM

The LSTM model performed decently and showed better results than the Simple RNN model. The highest accuracy value (0.88) was observed in the original data, resulting in high precision in positive class and recall in negative. With data without stopwords the efficiency dropped, and lemmatization slightly improved the metrics. The highest AUC-ROC appeared in original data as expected.

- CNN

CNN models performed well, comparable to Logistic Regression and SGD Classifier. In the original data the highest accuracy (0.9) was detected. The AUC-ROC consistently reached 0.94, reflecting its great performance across preprocessing methods.

- Overall Observations

The top performers were SGD Classifier and Logistic Regression. Lemmatization and stopwords removal didn't drastically affect any classifier, with Simple RNN benefiting the most from lemmatization. TF-IDF features slightly enhanced the performance of SGD Classifier and Multinomial Naïve Bayes. Multinomial Naïve Bayes, while maintaining strong precision for the positive class and recall for the negative, suffered in precision for the negative class and recall for the positive. CNN models provided comparable results with decent performance across all data forms. The LSTM model performed favorably in training and predicting sentiments, with the only negative element being the long computational time required.

Conclusion

By analyzing the results of the models, it was observed that the Machine Learning models outperformed the deep learning models in terms of sentiment analysis accuracy and efficiency. Logistic Regression and SGD Classifier stood out as top-performing models, achieving high accuracy, precision, recall, and F1 scores across different preprocessing methods and feature types. The complexity of deep learning models, such as LSTM and Simple RNN, did not necessarily improve the classification's efficiency or prediction accuracy. CNNs, however, showed promise by achieving comparable performance to Logistic Regression and SGD Classifier, with faster training times relative to other neural networks. The heaviest model computationally was the LSTM, which recorded the longest training times, exceeding at least a minute. Simple RNN followed in computational expense. Among neural networks, CNN demonstrated the fastest training times combined with decent predictive performance.

These findings suggest that while deep learning models hold potential for larger or more complex datasets, traditional Machine Learning approaches remain highly effective and computationally efficient for this sentiment analysis task.

Future Considerations

To increase the efficiency of Machine Learning models, hyperparameter tuning could be explored. As mentioned by Achmad Baroqah Pohan et al., (2024) [8], utilizing the GridSearchCV() method from scikit-learn allows trying different combinations of hyperparameters for specific datasets. While computationally expensive, it can enhance model precision significantly and is worth considering for optimization.

For deep neural networks and NLP classification tasks, adjustments can be made based on the dataset's complexity, text data size, language, and vocabulary. This includes increasing or decreasing the complexity of the model, adding more layers to address overfitting, and applying regularization techniques. Pretrained models like BERT, which are large language models, should also be considered, as they perform exceptionally well on sentiment analysis tasks due to their complete understanding of language and context.

Links

- Dataset: https://www.kaggle.com/datasets/harisyafie/baldurs-gate-3-steam-reviews
- Jupyter Notebook 1 (Machine Learning Models): https://www.kaggle.com/code/nicktsopanidis/baldur-s-gate-3-steam-reviews-sentiment-analysis
- Jupyter Notebook 2 (Deep Learning Models): https://www.kaggle.com/code/nicktsopanidis/bg3-reviews-sentiment-analysis-with-deep-learning

References

1. *Baldur's Gate 3 Steam Reviews*. (n.d.). Www.kaggle.com. https://www.kaggle.com/datasets/harisyafie/baldurs-gate-3-steam-reviews

2. Jahromi, A. H., & Taheri, M. (2017, October 1*). A non-parametric mixture of Gaussian naive Bayes classifiers based on local independent features*. IEEE Xplore. https://doi.org/10.1109/AISP.2017.8324083

3. GeeksforGeeks. (2023, November 17). *Multinomial Naive Bayes*. GeeksforGeeks. https://www.geeksforgeeks.org/multinomial-naive-bayes/

4. mehakiftikhar. (2024, March 7). *Sentiment Analysis | LSTM | Accuracy 96%*. Kaggle.com; Kaggle. https://www.kaggle.com/code/mehakiftikhar/sentiment-analysis-lstm-accuracy-96/notebook#Sentiment-Analysis-%7C-LSTM-%7C-Accuracy-96%25

5. Castillo, J. L. (2023, June 8). *Sentiment Analysis Using LSTM Networks: A Deep Dive into Textual Data. Medium*. https://javilopezcastillo.medium.com/sentiment-analysis-using-lstm-networks-a-deep-dive-into-textual-data-61cdd2e43dec

6. *Sentiment Analysis with an Recurrent Neural Networks (RNN).* (2022, October 7). GeeksforGeeks. https://www.geeksforgeeks.org/sentiment-analysis-with-an-recurrent-neural-networks-rnn/

7. Collins, E. R. (2024, March 8). *Using TensorFlow to Convert Tokenized Words from the Iliad Dataset into Integers in Python – Be on the Right Side of Change*. Finxter.com. https://blog.finxter.com/using-tensorflow-to-convert-tokenized-words-from-the-iliad-dataset-into-integers-in-python/

8. Achmad Baroqah Pohan, Irmawati, & Kurniasih, A. (2024). *Optimization of Classification Algorithm with GridSearchCV and Hyperparameter Tuning for Sentiment Analysis of the Nusantara Capital City*. Journal of Artificial Intelligence and Engineering Applications (JAIEA), 3(3), 808–814. https://doi.org/10.59934/jaiea.v3i3.514